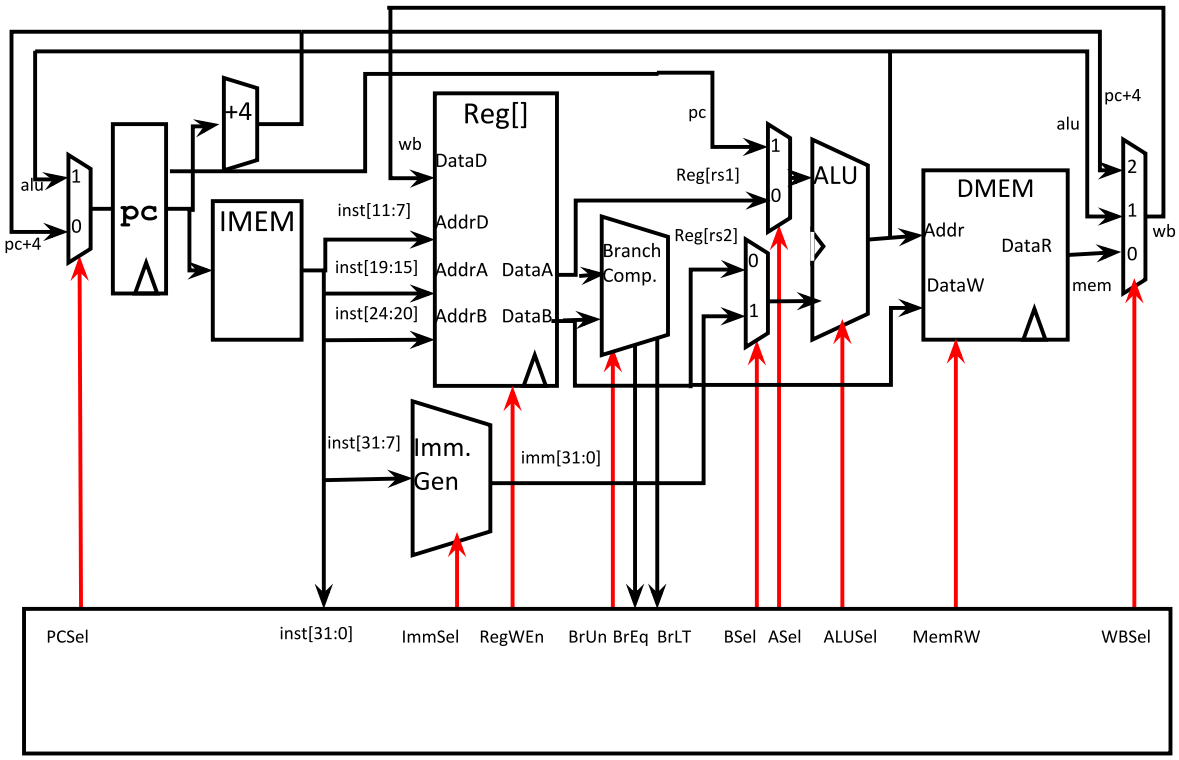
EECS 151/251A Homework 5

Due Friday , October 11th, 2019

Problem 1: Hazards

Consider the standard single-cycle RISC-V datapath and control unit:



a) The following proposed instructions require new hardware be added to the datapath. Draw the updated diagram to support the following instructions.

   i) `add rd, rs1, rs2, rs3:  rd = rs1 + rs2 + rs3`

   ii) `swadd rs1, rs2, imm:  M[rs1] = rs2 + imm`

b) Manufacturing defects could make certain circuits non-functional or can force signals to a particular value. For the following defects, select the instructions that would *not* work if the defect was present and explain why.

(a) Adder in the ALU doesn't work
- sll
- beq
- sw
- jal

(b) `RegWEn` is stuck at 0
- lw
- sw
- add
- beq

(c) `ASel` is stuck at 0
- jalr
- addi
- beq
- auipc

Problem 2: Pipelines

Assume the datapath is pipelined as such:

Instruction Fetch (IF)	Execute (EX)	Memory + Writeback (MWB)
Read from IMEM Generate immediate Decode instruction	Regfile read Branch comparison ALU operation	DMEM access Regfile writeback

a) No forwarding has been implemented. For the following assembly, finish out the pipeline table and find out how many cycles it takes to implement (the following table is just an example and is not drawn fully).

```
add x0, x1, x2
sub x3, x2, x4
add x0, x3, x4
or  x3, x2, x1
and x4, x1, x0
xor x2, x1, x4
add x1, x2, x0
```

Cycle	IF	EX	WB
1	add	-	-
2	sub	add	-

b) Assume the same instructions are implemented on a 5-stage CPU as shown in the following diagram. Redraw the pipeline diagram to add necessary components so ALU $->$ ALU forwarding is implemented, and redo the pipeline table. How many cycles does it take now?

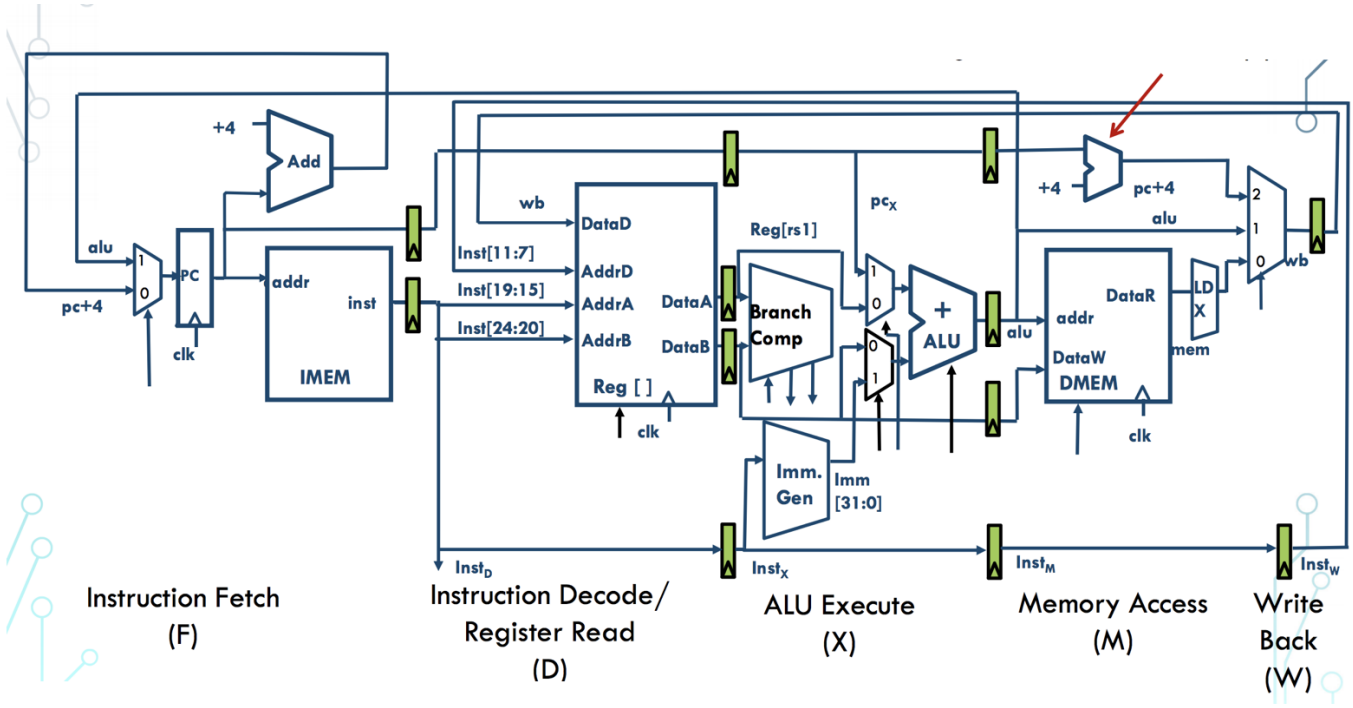


Figure 1: 5-stage pipeline

Problem 3: Branch Prediction

For the following assembly code implemented on the 5-stage pipeline diagram shown in the previous problem and ALU forwarding is implemented, fill out the pipeline table. Do this for the following two cases: 1) pc+4 is always taken 2) branch is always taken.

```
0x00:   li x0, 3
0x04:   li x1, 7
0x08:   add x2, x0, x1
0x0c:   li x3, 10
0x10:   bne x3, x2, 0x0c
0x14:   addi x0, x3, 5
0x18:   subi x3, x3, 5
0x1c:   addi x0, x3, 6
0x20:   nop
```